Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji
Akceleracja obliczeń w przetwarzaniu danych
Dokumentacja projektu

Symulacja N-ciał

Autorzy:
Jakub Misiurka 261428
Kinga Foksińska 255591
Michał Korczak 263923
Piotr Walerianowicz 264027

Grudzień 2024

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Głównym celem projektu jest porównanie wydajności i efektywności obliczeniowej metody 2-Pair oraz Barnes-Hut na różnych platformach sprzętowych: CPU oraz GPU. Metoda 2-Pair jest algorytmem obliczającym siły grawitacyjne między wszystkimi parami ciał w symulacji N-ciał, co stanowi problem o złożoności obliczeniowej $\mathcal{O}(N^2)$. Dzięki zastosowaniu technik równoległego przetwarzania projekt dąży do zbadania możliwości przyspieszenia tych obliczeń.

Oprócz tego, jako rozszerzenie projektu, zaimplementowano metodę Barnes-Hut na CPU oraz GPU. Jest to algorytm hierarchiczny, który redukuje liczbę obliczeń dzięki przybliżeniom w siłach grawitacyjnych, wykorzystując strukturę drzewa oktantowego.

1.2 Zakres projektu

Zakres projektu obejmuje:

1. **Implementację metody 2-Pair na CPU i GPU:** Zaimplementowano wersję metody 2-Pair dla jednostek centralnych (CPU) oraz procesorów graficznych (GPU). Obie wersje zostały zoptymalizowane pod kątem wydajności i dokładności.
2. **Porównanie wydajności CPU i GPU:** Przeprowadzono testy symulacyjne dla różnych wartości liczby ciał N, aby zmierzyć czas obliczeń oraz skalowalność algorytmu na CPU i GPU.
3. **Implementacja metody Barnes-Hut na CPU i GPU:** W ramach rozszerzenia projektu zaimplementowano metodę hierarchiczną *Barnes-Hut* na CPU oraz GPU.

Projekt kładzie główny nacisk na analizę wydajności metody 2-Pair Method oraz Barnes-Hut na CPU i GPU, co ma na celu ukazanie potencjału technologii równoległego przetwarzania w kontekście złożonych obliczeń fizycznych.

2 Podstawy teoretyczne

2.1 Problem N ciał

Problem N ciał to klasyczne zagadnienie w mechanice klasycznej i astrofizyce, które polega na przewidywaniu ruchu N ciał oddziałujących ze sobą pod wpływem sił grawitacyjnych. Równania ruchu opierają się na prawie powszechnego ciążenia Newtona, które opisuje siłę działającą między dwoma ciałami jako:

$$F_{ij} = G\frac{m_i m_j}{r_{ij}^2}\hat{r}_{ij},$$

gdzie F_{ij} to siła grawitacyjna między ciałami i i j, G jest stałą grawitacyjną, m_i i m_j to masy ciał, r_{ij} to odległość między nimi, a $\hat{r}_{ij}$ to jednostkowy wektor wskazujący kierunek siły.

Złożoność problemu wynika z faktu, że każda para ciał wpływa na siebie nawzajem, co prowadzi do konieczności obliczenia $\binom{N}{2}$ interakcji w każdej iteracji symulacji. Dla każdego ciała i, musimy obliczyć jego interakcję z ciałem j ($j > i$), a ponieważ interakcja $i \leftrightarrow j$ i $j \leftrightarrow i$ jest taka sama, uwzględniamy każdą parę tylko raz.

Przykład:

Jeśli $N = 4$ (4 ciała), liczba unikalnych par wynosi:

$$\binom{4}{2} = \frac{4 \cdot 3}{2} = 6$$

. Pary to: $(1,2)$, $(1,3)$, $(1,4)$, $(2,3)$, $(2,4)$, $(3,4)$.

Złożoność obliczeniowa algorytmu wynosi więc $\mathcal{O}(N^2)$, co dla dużych N staje się kluczowym wyzwaniem dla efektywnej symulacji.

2.2 Algorytmy

W ramach projektu zaimplementowano dwa algorytmy:

2.2.1 2-Pair Method

Metoda 2-Pair Method to prosty i dokładny algorytm, który oblicza siły grawitacyjne dla wszystkich par ciał w układzie. Każda iteracja algorytmu składa się z następujących kroków:

1. Obliczenie sił grawitacyjnych dla każdej pary ciał.
2. Aktualizacja prędkości ciał na podstawie uzyskanych sił.
3. Aktualizacja pozycji ciał w oparciu o nowe prędkości.

Algorytm ten jest używany zarówno na CPU, jak i GPU, co pozwala na bezpośrednie porównanie ich wydajności.

2.2.2 Barnes-Hut Method

Metoda Barnes-Hut opiera się na założeniu, że wpływ sił grawitacyjnych grupy ciał na dane ciało może być przybliżony, jeśli odległość między tym ciałem a grupą jest wystarczająco duża. Kluczowym elementem algorytmu jest podział przestrzeni symulacji na hierarchiczne regiony (oktanty). Podstawowym narzędziem w algorytmie jest drzewo ósemkowe (octree). Każdy węzeł drzewa reprezentuje region przestrzeni, a jego dzieci odpowiadają podziałowi tego regionu na mniejsze części:

1. **Węzły liści** przechowują pojedyncze ciała znajdujące się w danym regionie.
2. **Węzły wewnętrzne** zawierają informacje o grupie ciał, w tym ich łącznej masie oraz środku masy.

Budowa drzewa rozpoczyna się od korzenia reprezentującego całą przestrzeń symulacji. Następnie, dla każdego ciała, określa się odpowiedni podregion, aż do osiągnięcia węzła liścia lub węzła zawierającego tylko jedno ciało.

Dla każdego ciała siły grawitacyjne są obliczane na podstawie interakcji z innymi ciałami oraz regionami przestrzeni reprezentowanymi przez węzły drzewa. Decyzja, czy uwzględnić cały węzeł, czy jego podregiony, opiera się na kryterium θ:

$$\frac{s}{d} < \theta,$$

gdzie s to rozmiar regionu reprezentowanego przez węzeł, a d to odległość między badanym ciałem a środkiem masy tego węzła.

1. Jeśli warunek jest spełniony, wpływ całego węzła jest traktowany jako skoncentrowany w jego środku masy.
2. Jeśli warunek nie jest spełniony, siły są obliczane rekurencyjnie dla podregionów tego węzła.

Proces podziału przestrzeni na mniejsze regiony pozwala na zgrupowanie ciał w odległych częściach przestrzeni w jednym węźle. Dzięki temu liczba bezpośrednich obliczeń sił jest znacząco zmniejszona, szczególnie w przypadku dużych zbiorów danych.

Przybliżenie stosowane w algorytmie polega na zastąpieniu grup ciał w odległych regionach jednym punktem masowym. Umożliwia to znaczącą redukcję liczby obliczeń, przy jednoczesnym zachowaniu akceptowalnej dokładności wyników. Dokładność ta zależy od wyboru parametru θ, który ustala kompromis między szybkością obliczeń a precyzją.

2.3 Architektura obliczeniowa

2.3.1 Podstawowe różnice między CPU a GPU w kontekście równoległego przetwarzania

CPU (Central Processing Unit) to uniwersalny procesor zaprojektowany do wydajnego wykonywania zróżnicowanych, sekwencyjnych zadań. Z kolei GPU (Graphics Processing Unit) to procesor zoptymalizowany do równoległego przetwarzania wielu danych, pierwotnie stworzony do obsługi grafiki komputerowej, a obecnie szeroko stosowany w obliczeniach naukowych.

Kluczowe różnice:

1. **Architektura:** CPU posiada kilka rdzeni o wysokiej wydajności, zoptymalizowanych do obsługi wątków sekwencyjnych. GPU dysponuje tysiącami rdzeni, które mogą równolegle przetwarzać duże ilości danych.
2. **Równoległość:** GPU oferuje znacznie większy stopień równoległości, co czyni je idealnym dla problemów obliczeniowych wymagających obsługi dużych macierzy lub wielu iteracji tego samego algorytmu.
3. **Przepustowość pamięci:** GPU ma wyższą przepustowość pamięci, co pozwala na szybszy transfer danych, ale większe opóźnienia w dostępie do pamięci mogą ograniczać wydajność w aplikacjach zależnych od danych.

2.3.2 Zalety i ograniczenia CPU i GPU dla problemu N ciał

Zalety CPU

1. Łatwość implementacji algorytmów, które są trudne do zrównoleglenia.
2. Uniwersalność i możliwość wykonywania różnych typów obliczeń w jednym programie.
3. Optymalizacja dla algorytmów wymagających małej ilości wątków.

Ograniczenia CPU

1. Ograniczona liczba rdzeni ogranicza równoległość.
2. Niska przepustowość pamięci w porównaniu z GPU.

Zalety GPU

1. Wysoka wydajność w obliczeniach równoległych.
2. Zdolność do obsługi dużych zestawów danych w krótkim czasie.
3. Efektywność dla algorytmów o złożoności $\mathcal{O}(N^2)$, takich jak 2-Pair Method.

Ograniczenia GPU

1. Trudność w implementacji i konieczność precyzyjnego zarządzania pamięcią.
2. Zależność od przepustowości pamięci i czasochłonność transferu danych między CPU a GPU.
3. Mniej efektywne działanie dla algorytmów silnie zależnych od danych lub trudnych do zrównoleglenia.

3 Implementacja

3.1 Implementacja 2-pair method

3.1.1 Implementacja na CPU

Implementacja 2-pair method na procesorach (CPU) została zoptymalizowana do efektywnego wykorzystania wielu rdzeni procesora za pomocą biblioteki OpenMP, która umożliwia równoległe przetwarzanie obliczeń. Algorytm ten oblicza wzajemne oddziaływania grawitacyjne między ciałami w przestrzeni 3D, wykorzystując zasadę symetrii sił grawitacyjnych.

Struktura projektu

Kod został podzielony na kilka modułów:

1. **main.cpp** – Główny plik programu, inicjalizujący dane wejściowe oraz wywołujący funkcje obliczeniowe i zapisujące wyniki.
2. **physics.cpp** – Zawiera implementację fizyki symulacji, w tym obliczenia sił, aktualizację prędkości i pozycji oraz funkcje zapisu wyników.
3. **physics.h** – Plik nagłówkowy definiujący strukturę danych (Body) oraz deklaracje funkcji fizycznych.

Główne komponenty

1. **Struktura Body**
   Reprezentuje ciała w symulacji:
   (a) pozycja,
   (b) prędkość,
   (c) masa,
   (d) funkcja resize - dostosowuje rozmiar wektorów do liczby ciał,
   (e) funkcja to_json - eksportuje dane ciała do formatu JSON.
2. **Funkcja update_velocities**
   Aktualizuje prędkości ciał zgodnie z prawem grawitacji Newtona:
   (a) Iteruje przez każdą parę ciał.
   (b) Oblicza odległość między ciałami i siłę grawitacyjną:

$$F_{ij} = G\frac{m_i m_j}{r_{ij}^2}\hat{r}_{ij},$$

   Obliczenia dla pary i,j są wykonywane raz, co zmniejsza liczbę iteracji wewnętrznej pętli o połowę.

Dodatkowo zastosowano dodanie 1e-9 do odległości, czyli dodanie dystansu na tyle małego, że nie będzie miał wpływu na otrzymane wyniki. To podejście eliminuje ryzyko dzielenia przez zero, które mogłoby wystąpić w przypadku bardzo bliskich ciał.

(c) Aktualizuje prędkości:

i. Dla ciała i: $v_x^i += \frac{F_x}{m_i} \cdot \Delta t$.

ii. Dla ciała j: $v_x^j -= \frac{F_x}{m_j} \cdot \Delta t$.

3. **Funkcja update_positions**
Aktualizuje pozycje ciał w przestrzeni 3D. Nowa pozycja obliczana jako: $x = x + v_x \cdot \Delta t$

Wykorzystanie OpenMP

Do równoległego przetwarzania obliczeń sił i aktualizacji pozycji ciał zastosowano dyrektywy OpenMP:

1. **Równoległe pętle dla obliczeń sił i pozycji**
Pętle iterujące przez wszystkie pary ciał (i, j) zostały równoległe dzięki dyrektywie `#pragma omp parallel for`, co pozwala podzielić obliczenia między dostępne rdzenie procesora:

```
#pragma omp parallel for schedule(dynamic, 64)
```

Parametr `schedule(dynamic, 64)` dystrybuuje iteracje zewnętrznej pętli w porcjach po 64 iteracje na wątek. Dynamiczne przydzielanie zapewnia, że wątki, które skończą swoje porcje wcześniej, dostaną kolejne, co minimalizuje nierównomierność obciążenia.

2. **Atomowe operacje**
W przypadkach, gdzie wiele wątków mogłoby jednocześnie modyfikować wspólne dane, zastosowano dyrektywę `#pragma omp atomic`, aby zapewnić bezpieczeństwo wątków, np. podczas aktualizacji prędkości ciał.
Dlaczego nie critical? Dyrektywa atomic jest bardziej wydajna niż critical, ponieważ synchronizuje tylko pojedyncze operacje na danych.

3.1.2 Implementacja na GPU

Implementacja metody 2-pair na GPU została zoptymalizowana pod kątem równoległego przetwarzania z wykorzystaniem modelu programowania CUDA. Dzięki zastosowaniu GPU możliwe jest znaczne przyspieszenie obliczeń poprzez równoczesne przetwarzanie dużej liczby interakcji między ciałami. Podobnie jak w wersji na CPU, metoda opiera się na zasadzie symetrii sił grawitacyjnych.

Struktura projektu

Kod źródłowy został podzielony na trzy główne moduły:

1. **main.cu** – Główny plik programu, w którym znajdują się funkcje obliczeniowe na GPU oraz pętla symulacji.

2. **initialization.cu** – Odpowiada za inicjalizację pozycji, prędkości, mas i przyspieszeń ciał w przestrzeni 3D, z losowym rozkładem pozycji.

3. **file_operations.cu** – Zajmuje się zapisem wyników symulacji w formacie JSON, z możliwością dodawania nowych danych do istniejącego pliku.

Główne komponenty

1. **Struktura Bodies**
   Reprezentuje ciała w symulacji:
   (a) pozycja,
   (b) prędkość,
   (c) przyspieszenie,
   (d) masa.
2. **Funkcja initializeBodies**
   Wykorzystuje moduł initialization.cu do losowego rozmieszczenia ciał w przestrzeni, z ustaloną masą dla każdego z nich.
3. **Obliczenia na GPU:**
   (a) **Kernela computeBodyAcceleration**
   Oblicza przyspieszenia każdego ciała na podstawie wzajemnych oddziaływań grawitacyjnych:
   $$\vec{a}_i = \sum_{j \neq i} G \frac{m_j}{|\vec{r}_j - \vec{r}_i|^3} (\vec{r}_j - \vec{r}_i),$$
   gdzie dodano mały dystans (0.01) w celu uniknięcia dzielenia przez zero. Każdy wątek w GPU odpowiada za jedno ciało, co pozwala na równoległe przetwarzanie.
   (b) **Kernela computeNewPositionAndSpeed**
   Aktualizuje pozycje i prędkości ciał zgodnie z równaniami:
   $$\vec{v}_i = \vec{v}_i + \vec{a}_i \cdot \Delta t,$$
   $$\vec{r}_i = \vec{r}_i + \vec{v}_i \cdot \Delta t.$$

Zastosowanie CUDA

1. **Wykorzystanie równoległości**
   Każda iteracja obliczeń została podzielona na bloki i wątki, dzięki czemu GPU może równolegle obliczać interakcje grawitacyjne dla wszystkich ciał. Rozmiar bloku został ustalony na 256 wątków (`#define BLOCK_SIZE 256`), a liczba bloków (`numberOfBlocks`) zależy od liczby ciał (N).
2. **Efektywne zarządzanie pamięcią**
   Dane ciał są przechowywane w ciągłym bloku pamięci GPU, co zmniejsza narzut na przesyłanie danych między CPU a GPU.
3. **Synchronizacja urządzenia**
   Po wykonaniu każdej kerneli wywoływana jest funkcja `cudaDeviceSynchronize`, aby zapewnić, że wszystkie obliczenia zostały zakończone przed przejściem do następnej operacji.

3.2 Implementacja algorytmu Barnes-Huta

3.3 Implementacja na CPU

Metoda Barnes-Hut została zaimplementowana w celu efektywnego modelowania oddziaływań grawitacyjnych między ciałami w przestrzeni 3D. Używa ona struktury drzewa oktantowego (Barnes-Hut Tree), która pozwala zredukować złożoność obliczeniową symulacji z $O(N^2)$ do $O(N \log N)$. Optymalizacja wykorzystuje równoległość procesorów CPU przy użyciu biblioteki OpenMP.

Struktura projektu

Kod został podzielony na kilka modułów, co zapewnia czytelność i łatwość zarządzania:

1. **main.cpp** – Główna funkcja programu. Inicjalizuje ciała, przeprowadza iteracje symulacji i zapisuje wyniki.

2. **Body.h, Body.cpp** – Reprezentuje pojedyncze ciało w symulacji, definiując jego masę, pozycję, prędkość oraz przyspieszenie.

3. **Octant.h, Octant.cpp** – Definiuje podregiony przestrzeni w drzewie Barnes-Hut. Umożliwia podział na 8 mniejszych oktantów i sprawdza, czy ciało należy do danego regionu.

4. **BHTreeNode.h, BHTreeNode.cpp** – Reprezentuje węzły drzewa Barnes-Hut. Zarządza wstawianiem ciał, obliczaniem środka masy, sił grawitacyjnych oraz rekursyjnym podziałem przestrzeni.

5. **Simulation.h, Simulation.cpp** – Obsługuje główne funkcje symulacyjne, takie jak aktualizacja pozycji ciał oraz budowa drzewa Barnes-Hut.

Główne komponenty

1. **Struktura Body**
   Modeluje każde ciało w przestrzeni z następującymi właściwościami:

   (a) masa,

   (b) pozycja,

   (c) prędkość,

   (d) przyspieszenie.

   Aktualizacja pozycji i prędkości odbywa się za pomocą metody Leapfrog.

2. **Metoda Leapfrog** Pozycja i prędkość każdego ciała są aktualizowane za pomocą metody Leapfrog, która jest jedną z najczęściej stosowanych metod w symulacjach N-ciał. Jest to metoda numeryczna drugiego rzędu, zapewniająca stabliność przy długoterminowych obliczeniach. Aktualiozacja w schemacie Leapfrog przebiega w dwóch etapach:

   (a) **Aktualizacja prędkości w połowie kroku czasowego($t + \frac{\Delta t}{2}$):**

   $$v_x = v_x + \frac{a_x \cdot \Delta t}{2}, \quad v_y = v_y + \frac{a_y \cdot \Delta t}{2}, \quad v_z = v_z + \frac{a_z \cdot \Delta t}{2}.$$

   (b) **Aktualizacja pozycji ($t + \Delta t$):**

   $$x = x + v_x \cdot \Delta t, \quad y = y + v_y \cdot \Delta t, \quad z = z + v_z \cdot \Delta t.$$

   (c) **Dokończenie aktualizacji prędkości:**

   $$v_x = v_x + \frac{a_x \cdot \Delta t}{2}, \quad v_y = v_y + \frac{a_y \cdot \Delta t}{2}, \quad v_z = v_z + \frac{a_z \cdot \Delta t}{2}.$$

   Metoda ta pozwala uniknąć strat energii, które mogą występować w mniej zaawansowanych schematach, takich jak metoda Eulera.

3. **Drzewo Barnes-Hut (BHTreeNode)**
   Reprezentuje węzły drzewa oktantowego i zarządza:

   (a) **insert** – dodaje ciało do odpowiedniego węzła, dzieląc przestrzeń na oktanty w razie potrzeby.

(b) **calculateForce** – oblicza siły grawitacyjne, stosując przybliżenie Barnes-Hut:

$$\text{Jeśli } \frac{\text{rozmiar regionu}}{\text{odległość}} < \theta,$$

region traktowany jest jako jeden punkt masy.

(c) **updateMassAndCenter** – aktualizuje masę i środek masy węzła.

(d) **subdivide** – dzieli przestrzeń na osiem mniejszych oktantów.

4. **Metoda symulacyjna**
Symulacja przebiega w następujących krokach:

(a) Budowa drzewa Barnes-Hut na podstawie pozycji ciał w przestrzeni. Region obejmujący ciała definiowany jest dynamicznie.

(b) Obliczanie sił grawitacyjnych na każde ciało, przy czym węzły spełniające warunek Barnes-Hut traktowane są jako pojedyncze punkty masy.

(c) Aktualizacja pozycji i prędkości ciał metodą Leapfrog.

Wykorzystanie OpenMP

1. **Równoległe obliczanie sił**
Pętle obliczające siły działające na każde ciało są równoległe:

```
#pragma omp parallel for
for (int i = 0; i < bodies.size(); ++i) {
    double fx = 0.0, fy = 0.0, fz = 0.0;
    root.calculateForce(bodies[i], fx, fy, fz);
    update_body_leapfrog(bodies[i], fx, fy, fz);
}
```

Każdy wątek przetwarza siły dla innego ciała, co znacznie przyspiesza obliczenia.

2. **Bezpieczeństwo wątków**
Dzięki lokalnym zmiennym (f_x, f_y, f_z) obliczenia w każdym wątku są niezależne, co eliminuje konieczność synchronizacji.

3.4 Implementacja na GPU

Metoda Barnesa-Huta została zaimplementowana w sposób wielowątkowy, z uwzględnieniem wykrycia wielkości bloku GPU przez program przy starcie, co pozwala na zmaksymalizowanie wydajności na dostępnym sprzęcie.

Kernele:

1. **Operacje atomowe** - funkcje atomicMaxDouble i *atomicMinDouble* zapewniają atomowe obliczanie maksimum i minimum dwóch liczb.

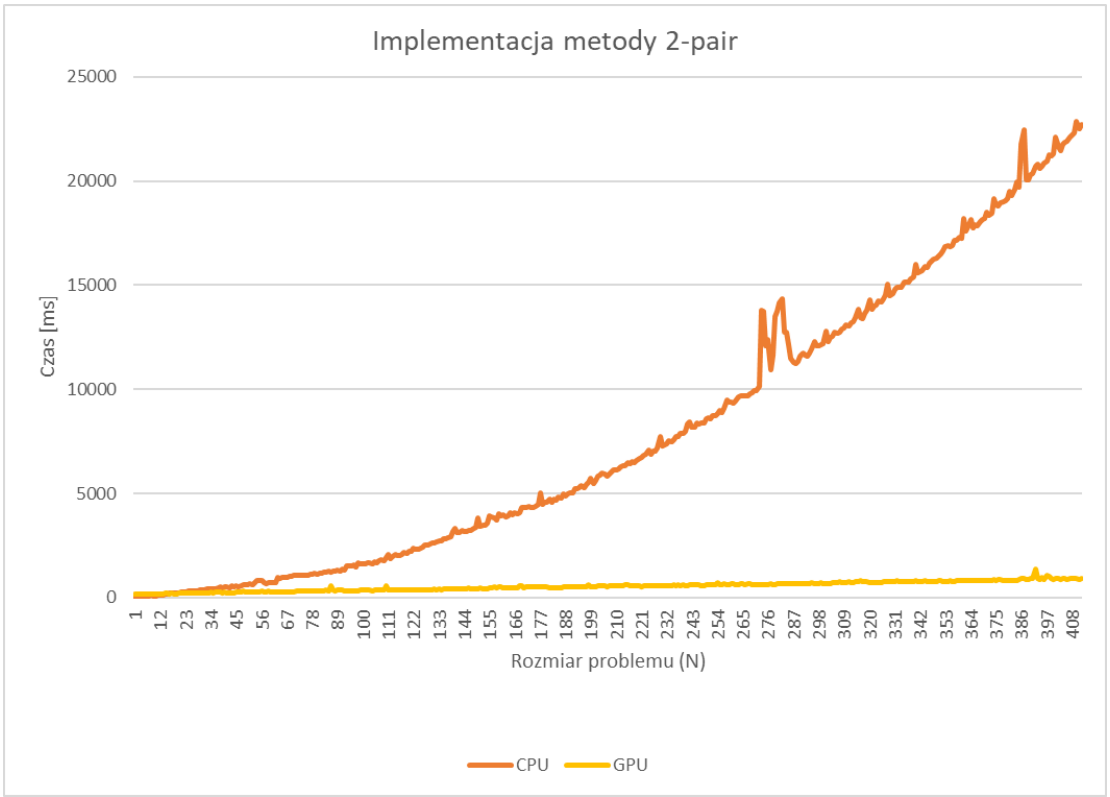
2. **computeBoundingBox** - oblicza maksymalne i minimalne współrzędne w danym zbiorze ciał *bodies*. Do porównania funkcja korzysta z metody turniejowej, która jest łatwa do zaimplementowania równolegle - w każdym kroku porównywane jest X liczb w $\frac{X}{2}$ parach, po czym w kolejnej iteracji $\frac{X}{2}$ wybranych liczb porównywane jest w $\frac{X}{4}$ parach aż do zmniejszenia par do $\frac{X}{X}$. W każdej iteracji bierze udział coraz mniej wątków, ale przy odpowiednio dużej liczbie wątków każda iteracja może być wykonywana w pełni równolegle. Metoda nazywana jest także metodą redukcji.

3. **insertBody** - funkcja wstawia do każdego węzła drzewa indeks przetwarzanego ciała, jeśli nie ma jeszcze przypisanego ciała, lub tworzy nowy węzeł w pozycji obliczonej na podstawie pozycji ciała.
4. **buildTree** - nakładka wywołująca *insertBody*.
5. **computeCenterOfMass** - funkcja obliczająca środek masy węzła. Dla liści drzewa jest to pozycja przypisanego do niego ciała, dla pozostałych węzłów średnia ważona mas jego potomków.
6. **computeForces i computeForcesRecursively** - funkcje obliczające siły działające na ciało. Na podstawie odległości pomiędzy węzłami podejmowana jest decyzja, czy obliczenia będą wykonane precyzyjnie, uwzględniając wszystkich potomków, czy jedynie środek masy węzła będzie uwzględniony.
7. **updateBodies** - funkcja aktualizująca pozycje, prędkości i przyspieszenia ciał.

4 Wyniki i analiza wydajności

4.1 Porównanie metody 2-pair na CPU i GPU

Przeprowadzono testy wydajności implementacji metody 2-pair dla symulacji N-ciał zarówno na CPU, jak i GPU. Celem było porównanie czasu obliczeń w zależności od rozmiaru problemu. Wyniki zaprezentowano na wykresie poniżej:



Rysunek 4.1: Wykres zależności rozmiaru problemu od czasu dla implementcji metody 2-pair na CPU i GPU

Wyniki pokazują znaczące różnice w wydajności między implementacją na CPU a GPU:

1. **CPU**: Czas obliczeń rośnie kwadratowo wraz ze wzrostem rozmiaru problemu. Wynika to z konieczności przeprowadzenia obliczeń dla wszystkich par ciał ($O(N^2)$). Dla dużych wartości N, czas obliczeń przekracza 20 s, co wskazuje na ograniczenia tej implementacji przy większej liczbie ciał.

2. **GPU**: Czas obliczeń jest niemal stały dla szerokiego zakresu wartości N. Wykorzystanie GPU pozwala na równoległe przetwarzanie, co skutkuje znaczącym przyspieszeniem. Nawet dla największych testowanych wartości N, czas obliczeń pozostaje na poziomie poniżej 500 ms.

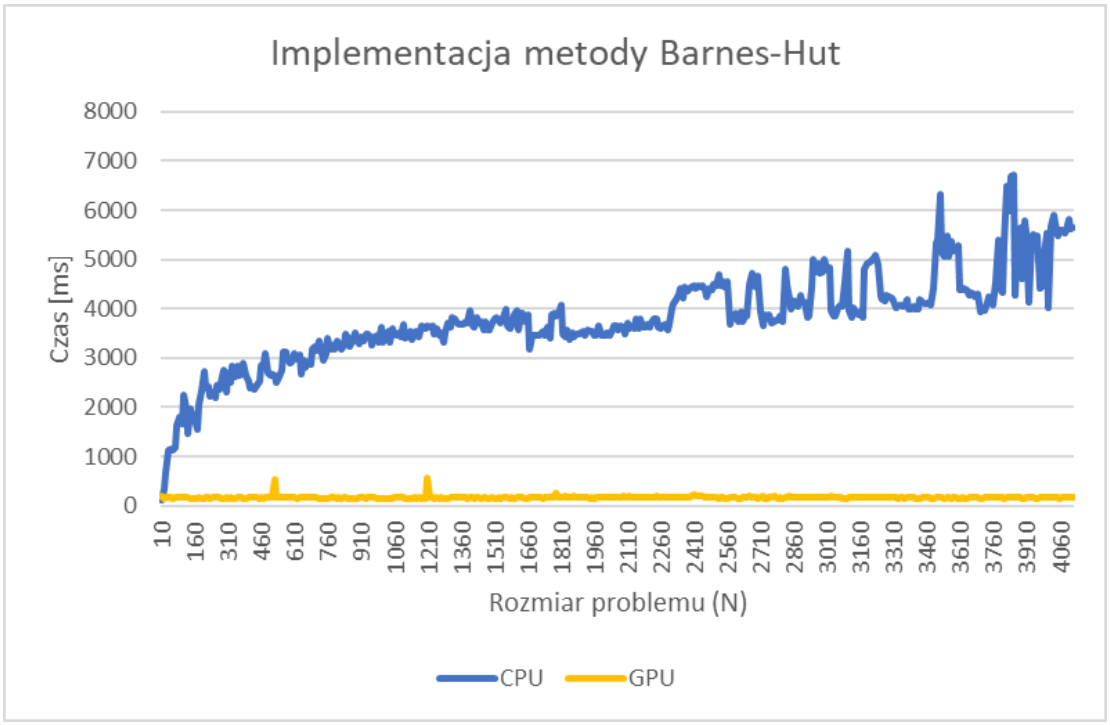
Implementacja GPU osiąga przyspieszenie o rzędy wielkości w porównaniu z wersją CPU. Wzrost efektywności wynika głównie z:

1. równoległego wykonywania obliczeń przy użyciu tysięcy wątków GPU,

2. redukcji narzutu wynikającego z ograniczeń pamięci cache, co jest kluczowe przy dużej liczbie ciał.

4.2 Porównanie metody Barnes-Hut na CPU i GPU

Wykres poniżej przedstawia porównanie czasów wykonania metody Barnes-Hut dla różnych rozmiarów problemu N na CPU oraz GPU.



Rysunek 4.2: Wykres zależności rozmiaru problemu od czasu dla implementcji metody Barnes-Hut na CPU i GPU

1. **Wydajność na CPU:** Metoda Barnes-Hut na CPU wykazuje stabilny wzrost czasu obliczeń wraz ze wzrostem liczby ciał. Przy mniejszych wartościach N czas obliczeń jest

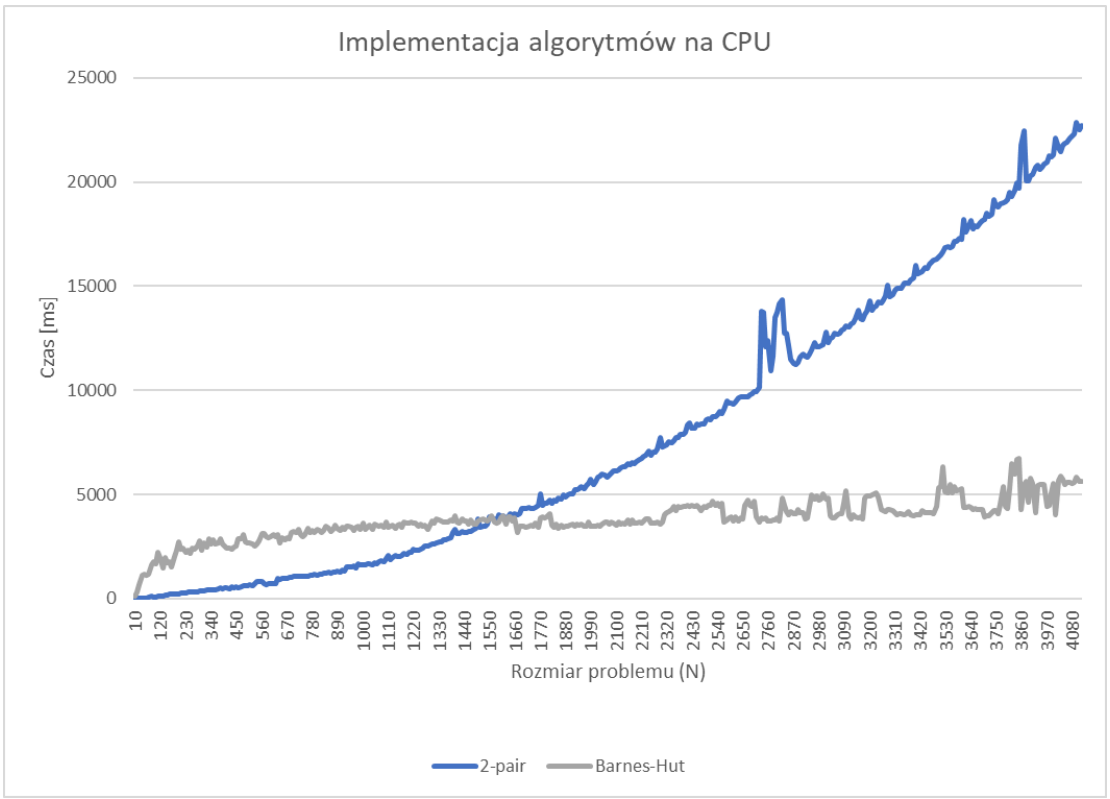
stosunkowo niski, jednak dla $N > 1000$ obserwuje się stopniowe wypłaszczenie wynikające z optymalizacji algorytmu $O(N \log N)$.

2. **Wydajność na GPU:** Implementacja GPU pokazuje znacznie krótsze czasy obliczeń w porównaniu do CPU. Wynika to z efektywnego wykorzystania równoległości w procesie obliczania sił oraz budowy drzewa. Czas obliczeń na GPU jest praktycznie stały w całym zakresie badanych wartości , co świadczy o jego znacznej skalowalności.

3. **Porównanie CPU i GPU:** Przy dużych wartościach N, GPU jest nieporównywalnie szybsze niż CPU. Dla $N = 4000$, czas obliczeń na GPU wynosi zaledwie ułamek czasu wymaganego na CPU. Niemniej jednak, efektywność GPU dla małych wartości N jest ograniczona przez narzut związany z transferem danych oraz inicjalizacją obliczeń równoległych, co powoduje niewielkie różnice w wydajności w początkowych punktach wykresu.

4.3 Porównanie metody 2-pair i Barnes-Hut na CPU

Przeprowadzono porównanie dwóch metod obliczeniowych, 2-pair oraz Barnes-Hut, dla symulacji N-ciał na CPU. Wyniki zilustrowano na poniższym wykresie:



Rysunek 4.3: Wykres zależności rozmiaru problemu od czasu dla implementcji na CPU

Wyniki pokazują wyraźne różnice w wydajności między obiema metodami:

1. **Metoda 2-pair**: Czas obliczeń rośnie kwadratowo wraz ze wzrostem rozmiaru problemu, co jest zgodne z teoretyczną złożonością obliczeniową $O(N^2)$. Dla dużych wartości N, czas obliczeń przekracza 20 s, co czyni tę metodę niewydajną przy większej liczbie ciał.

2. **Metoda Barnes-Hut**: Charakteryzuje się znacznie niższym czasem obliczeń dla dużych wartości N. Złożoność obliczeniowa tej metody wynosi $O(N \log N)$, co pozwala na efektywniejsze skalowanie w porównaniu z metodą 2-pair. Dla większych wartości N, różnica czasów między metodami staje się coraz bardziej znacząca.

Metoda Barnes-Hut wykazuje wyraźną przewagę nad metodą 2-pair w przypadku dużych symulacji. Jej przewaga wynika z zastosowania hierarchicznego podziału przestrzeni:

1. Dla mniejszych wartości N, różnica w czasie obliczeń między metodami jest mniej widoczna, ponieważ narzut związany z budową drzewa w Barnes-Hut może dominować.

2. Dla dużych wartości N, metoda Barnes-Hut oferuje znacznie lepszą skalowalność dzięki redukcji liczby obliczeń wymaganych do wyznaczenia sił między ciałami.

4.4 Wnioski

Przeprowadzone analizy i testy wydajności metod symulacji N-ciał pozwoliły na wyciągnięcie następujących wniosków:

1. Metoda **2-pair** jest prosta w implementacji, lecz jej złożoność obliczeniowa $O(N^2)$ czyni ją niewydajną dla dużych rozmiarów problemu. Czas obliczeń wzrasta wykładniczo wraz z liczbą ciał, co ogranicza jej zastosowanie do niewielkich symulacji.

2. Metoda **Barnes-Hut**, dzięki wykorzystaniu hierarchicznego podziału przestrzeni, znacząco poprawia skalowalność obliczeń. Złożoność $O(N \log N)$ pozwala na przeprowadzenie symulacji dla znacznie większych wartości N w akceptowalnym czasie.

3. W testach na CPU metoda Barnes-Hut przewyższała wydajnością metodę 2-pair, szczególnie dla $N > 1000$. Wyniki te potwierdzają zasadność stosowania bardziej zaawansowanych technik, takich jak drzewa oktalne czy kwadrantowe, w celu poprawy efektywności obliczeń.

4. Implementacja obliczeń na GPU wykazała istotne przyspieszenie zarówno dla metody **2-pair**, jak i **Barnes-Hut**. Szczególnie w przypadku metody Barnes-Hut, GPU umożliwia obliczenia dla dużych rozmiarów problemu $N > 4000$ w czasie o rząd wielkości krótszym niż CPU.

5. Porównanie metod na GPU i CPU jednoznacznie wskazuje na przewagę obliczeń równoległych, szczególnie w przypadku dużych symulacji. Metoda Barnes-Hut na GPU osiąga niemal stały czas obliczeń przy wzrastającym N, co czyni ją doskonałym wyborem dla dużych i złożonych symulacji.